HarvardX Data Science program capstone project

MoviLens recommendation system

Vladimir Pedchenko

10/09/2021

Contents

1 Introduction

This is a report of HarvardX Data Science program capstone project (PH125.9x).

Goal of the project is to create a movie recommendation system using the MovieLens dataset. Recommender Systems (RSs) are software tools and techniques providing suggestions for items to be of use to a user. It can be used in streaming services such YouTube or Netflix or in web-shops, to suggest user items which he/she, most likely, would buy.

In this specific case, we will try to predict rating of specific movie by specific user.

Before building a model we have to perform Exploratory data analysis (EDA) and select metric for model estimation. Metric is defined by project goal definition: we have to reach root mean squared error (RMSE) lower than 0.86490. Thus, RMSE is our metric for this project. It can be calculated by equation:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2},$$

where N is size of test-set, $y_{u,i}$ is the true rating given by user u to movie i and $\hat{y}_{u,i}$ is the predicted rating given by user u to movie i. Root mean squared error (RMSE) is reported in the same units as the outcomes, which makes understanding what is large and what is small enough RMSE more intuitive.

Final RMSE estimation will be performed on the final hold-out validation test set, which we will not use for any other purposes, neither for training model nor for model selection.

2 Data preparation

Code bellow was provided by HarvardX. It downloads data and split it to two datasets: **edx** and **validation**. **Validation** data set will not be used in the code until the final validation of our selected and trained model. Data analysis, model selection/training will be performed on **edx** dataset.

```
# download data
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)


# name columns
colnames(movies) <- c("movieId", "title", "genres")


# Create Data Frame with movies

# If using R 3.6 or earlier, comment out this statement and use above:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

# Join with rating by movieID
movielens <- left_join(ratings, movies, by = "movieId")
```

```r
# slice of movielens dataset to edx and validation datasets
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Check datasets dimensions: dimensions of Edx dataset are 9000055, 6 and dimensions of validation dataset are 999999, 6

3 Exploratory data analysis

3.1 First look on dataset

```r
class(edx)
```

```
## [1] "data.table" "data.frame"
```

Class of our dataset is data.frame, we can work with this data class as is. Let's see on first 6 records in the dataset:

Table 1: Edx dataset first records

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

We see that the dataset contains records of each rating was done by user and some information about the movie. For example, first line shows that user with ID = 1 had rated movie with ID=122 by five stars. Date and time of the rating can be extracted from the timestamp and we have additional information about the movie such as it's title, combined with year or release and genres of the movie.
The rating is the numeric variable, so it can take any numeric value. But we need to check, if it is a case. Unique ratings we can find in the dataset:

```
##  [1] 5.0 3.0 2.0 4.0 4.5 3.5 1.0 1.5 2.5 0.5
```

Statistics of ratings distribution:

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   4.000   3.512   4.000   5.000
```

Plot of ratings distribution:

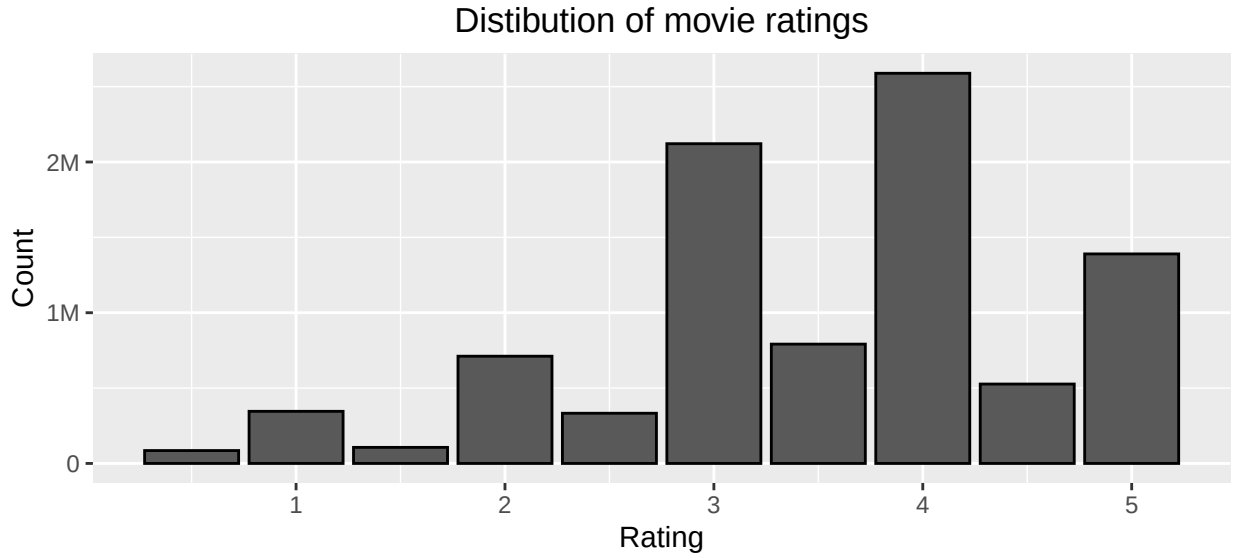

Figure 1: Distibution of movie ratings

If we consider ratings higher than average rating as "positive" and lower than average as "negative", we can find how many positive and negative ratings we have in our dataset:

Table 2: Negative vs. positive ratings

rating_type	n
negative	4494775
postitive	4505280

As we can see, numbers of positive and negative ratings are approximately equal.

Table 3: Half-star vs. whole-star ratings

rating_star	n
half_star	1843170
whole_star	7156885

After first look on the dataset we can conclude:

- Each row in the dataset represents single rating of user defined by userId column to movie, defined by movieId column, additional information about movie (genre and title) and rating timestamp;

- Ratings are in range 0.5 - 5.0, with 0.5 step;
- Whole-star rating is much more common than half-star;
- Average rating is about 3.5, but median is 4;
- The most common rating in the dataset is 4, and 50% of all ratings are lying between 3 and 4 (inclusive)

3.2 Movies ans users

Number of unique users in dataset: 69878; unique movies in dataset: 10677.

Total user/movie combinations amount should be: 746087406.

But as we saw before, we have only 9000055 records in the dataset. Only 1.2% of all possible combinations are rated.

To visualize this, we will sample 100 unique users and 100 unique movies and plot matrix with filled cells when user rated the movie and blank cells if not:

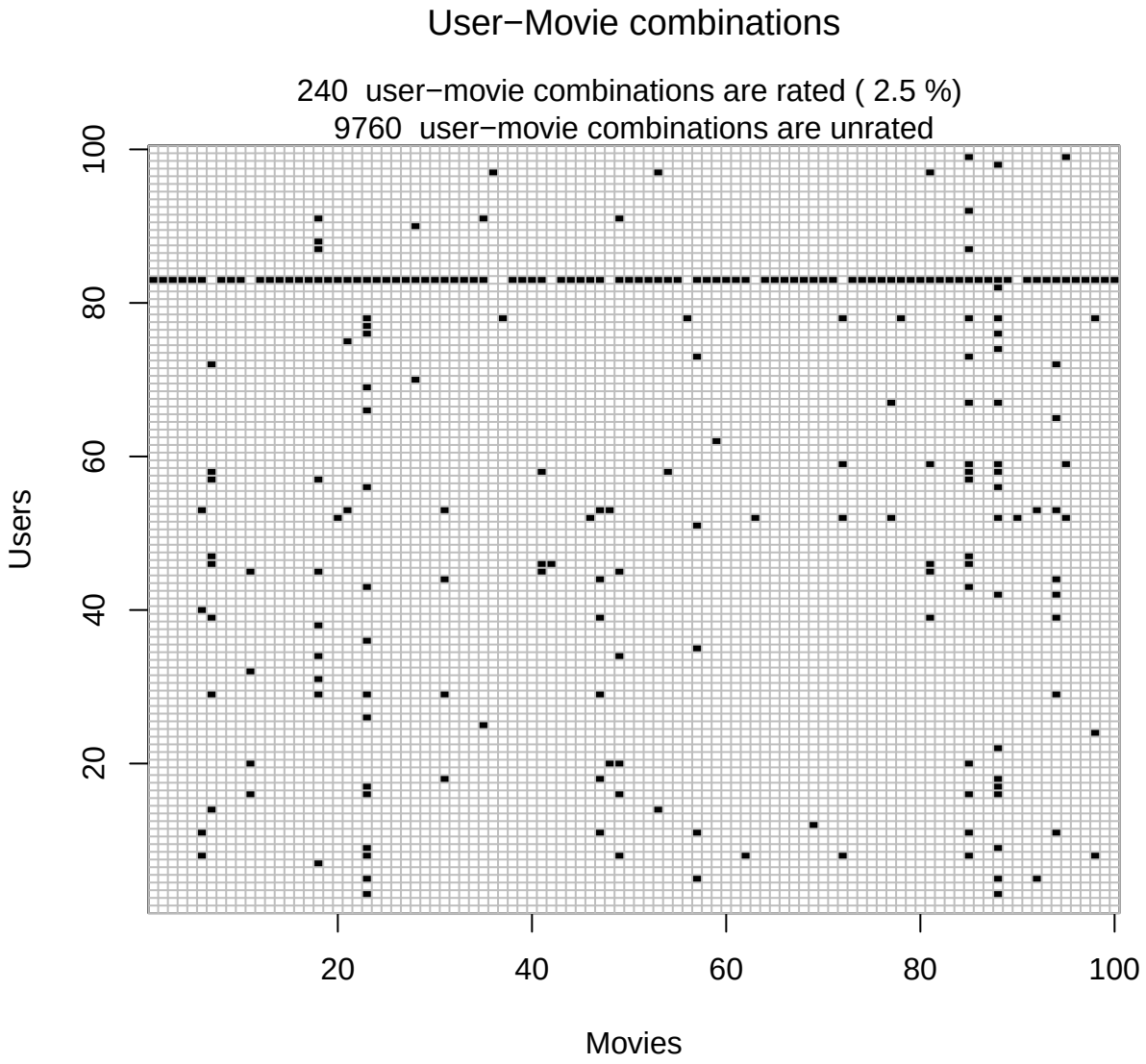

Figure 2: User-Movie combinations

Looking on Figure 2 we can rewrite our task: we have to build a model, which can fill the matrix for any given user and any given movie.

3.3 Movies analysis

First, let's look on the top-10 and bottom 10 movies:

Table 4: Best movies by rating

avg_rating	title
5.00	Hellhounds on My Trail (1999)
5.00	Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)
5.00	Shadows of Forgotten Ancestors (1964)

avg_rating	title
5.00	Fighting Elegy (Kenka erejii) (1966)
5.00	Sun Alley (Sonnenallee) (1999)
5.00	Blue Light, The (Das Blaue Licht) (1932)
4.75	Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)
4.75	Human Condition II, The (Ningen no joken II) (1959)
4.75	Human Condition III, The (Ningen no joken III) (1961)
4.75	Constantine's Sword (2007)

Table 5: Worst movies by rating

avg_rating	title
0.5000000	Besotted (2001)
0.5000000	Hi-Line, The (1999)
0.5000000	Accused (Anklaget) (2005)
0.5000000	Confessions of a Superhero (2007)
0.5000000	War of the Worlds 2: The Next Wave (2008)
0.7946429	SuperBabies: Baby Geniuses 2 (2004)
0.8214286	Hip Hop Witch, Da (2000)
0.8593750	Disaster Movie (2008)
0.9020101	From Justin to Kelly (2003)
1.0000000	Criminals (1996)

Looking on the tables, we see that the top-10 and bottom-10 movies are not widely known by it's title. Let's look on distribution of number of ratings of movies (how many times specific movie was rated):

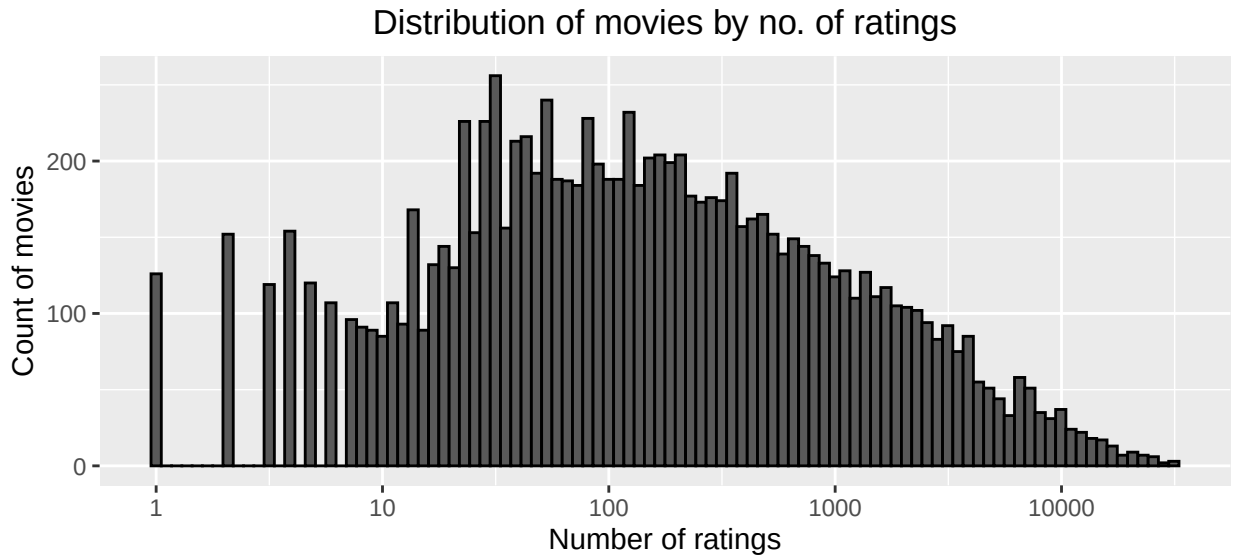

Figure 3: Distribution of movies by number of ratings

We can see, that approximately half of movies have less than 100 ratings and about 125 movies have only one rating. That can explain our observation of top/bottom 10 movies: very high and very low average movie rating can be done based on very few reviews, or even one review. This cannot be reliable and will be taken in account when building a prediction model.

To see, how different ratings are distributed across the dataset, we can plot distribution of the average ratings of movies:

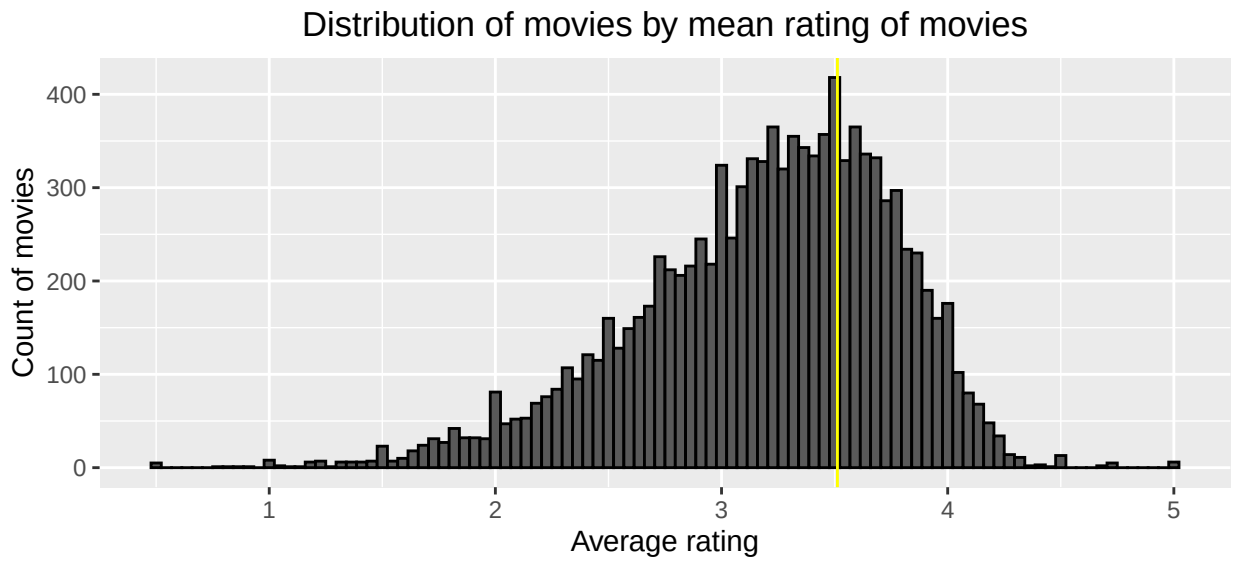

Figure 4: Distribution of movies by mean rating of movies

Thinking logically, the more ratings specific movie has, the more popular it is. Usually, good movies became very popular, therefore they should have higher average rating. To confirm or reject our hypotheses we can plot average movies ratings versus number of ratings for the movie:

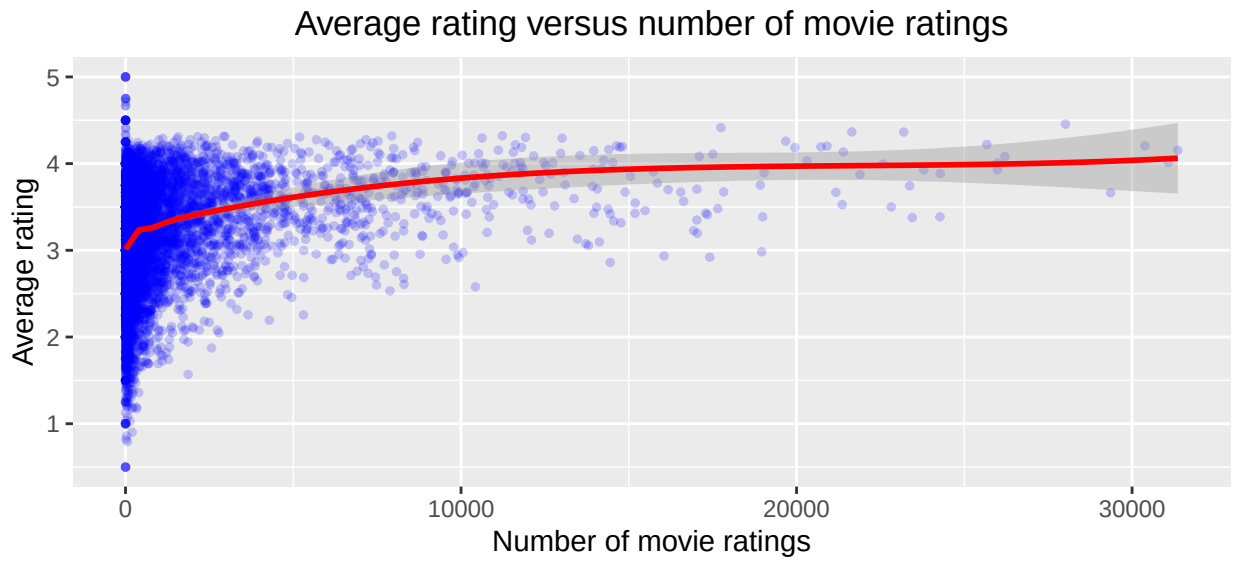

Figure 5: Average rating versus number of movie ratings

Our hypotheses is confirmed: more often rated movies are also have slightly higher average rating and smaller deviation.
Let's look at the top-10 and bottom-10 movies by number of ratings:

Table 6: Most popular movies

movieId	number_of_ratings	avg_rating	title
296	31362	4.154789	Pulp Fiction (1994)
356	31079	4.012822	Forrest Gump (1994)
593	30382	4.204101	Silence of the Lambs, The (1991)
480	29360	3.663522	Jurassic Park (1993)
318	28015	4.455131	Shawshank Redemption, The (1994)
110	26212	4.081852	Braveheart (1995)
457	25998	4.009155	Fugitive, The (1993)
589	25984	3.927859	Terminator 2: Judgment Day (1991)
260	25672	4.221311	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
150	24284	3.885789	Apollo 13 (1995)

Table 7: Least popular movies

movieId	number_of_ratings	avg_rating	title
3191	1	3.5	Quarry, The (1998)
3226	1	5.0	Hellhounds on My Trail (1999)
3234	1	3.0	Train Ride to Hollywood (1978)
3356	1	3.0	Condo Painting (2000)
3383	1	3.0	Big Fella (1937)
3561	1	1.0	Stacy's Knights (1982)
3583	1	3.0	Black Tights (1-2-3-4 ou Les Collants noirs) (1960)
4071	1	1.0	Dog Run (1996)
4075	1	1.0	Monkey's Tale, A (Les ChÃ¢teau des singes) (1999)
4820	1	2.0	Won't Anybody Listen? (2000)

Look at these tables confirms, that movies which were rated more often, have higher average rating. We can see that the movie "Hellhounds on My Trail (1999)" also appeared in Table 4: Best movies by rating, as it has average rating 5, but it is based only on a single rating, which cannot be reliable. We will take it in account during model building and tuning.

3.4 Users analysis

Now we will perform similar analysis but for users. First, let's look on the distribution of number of ratings for users:

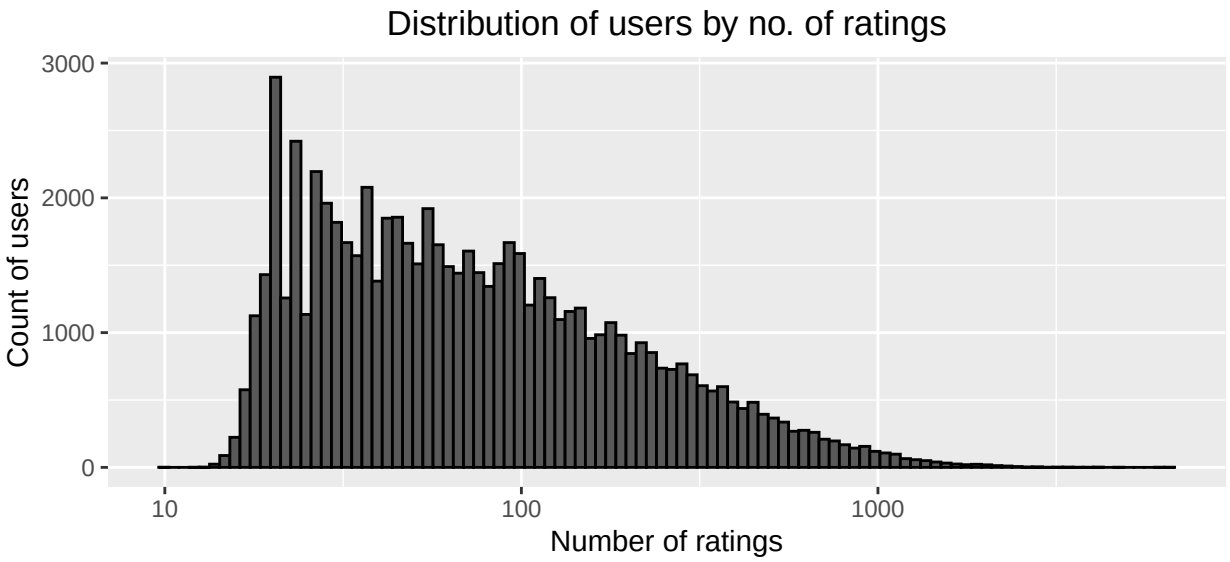

Figure 6: Distribution of users by number of ratings

Similar to number of ratings of movies, many users rated only few movies. We can see, that about half of users rated less than approximately 65 movies. We will also take it in account when building a model. To see, how different ratings are distributed across all users in the dataset, we can plot distribution of the average ratings which users give to movies:

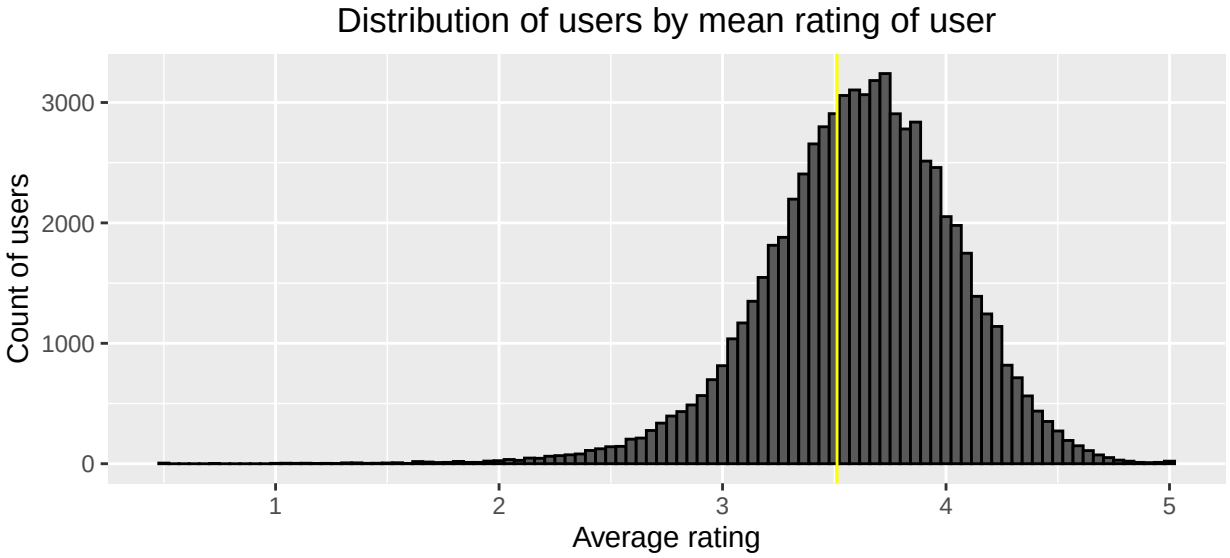

Figure 7: Distribution of users by mean rating of users

From the Figure 7 we can see, that some users tend to rate movies with higher score, unlike some of them prefer to give low rating. But majority of users have average rating given to movies higher than average; that makes sense: most of people prefer to watch movies of favorite genre, or with favorite actors, or movies which are blockbusters. In that case, the chance that user who selected specific movie for watching will like it, and rate with higher than average score is high.
Let's compare half-star ratings against whole-star rating again, but now for users:

Table 8: Half-star vs. whole-star ratings

rating_star	n
half_star	1843170
whole_star	7156885

To see, how number of movies rated by user effects on average rating by this user, we can plot one vs another (for users who rated at least 100 movies):

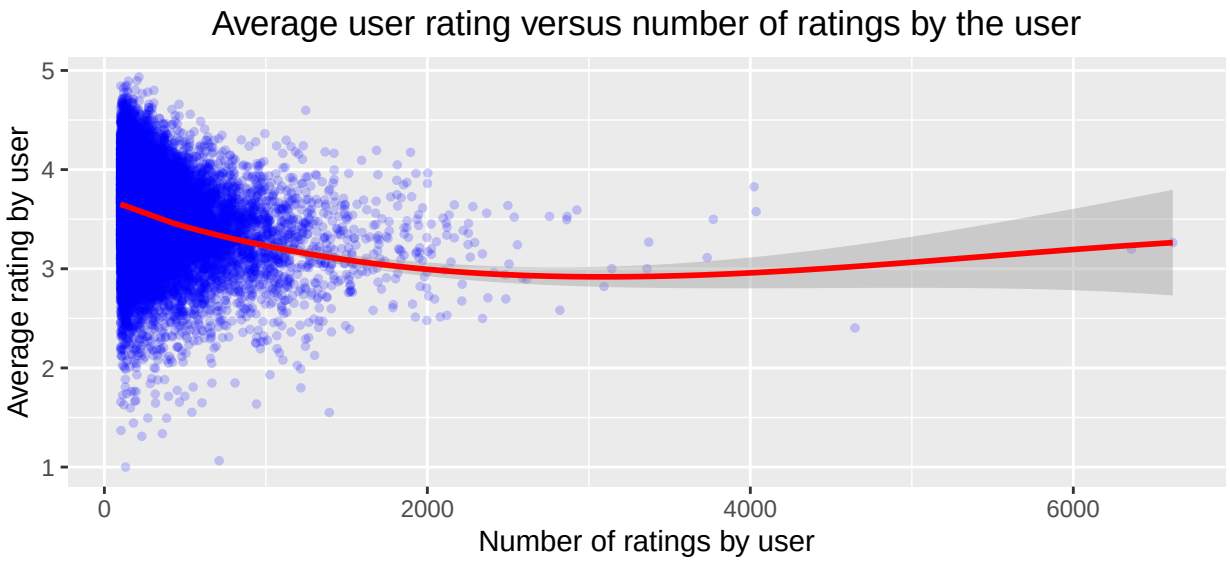

Figure 8: Average user rating versus number of user ratings

Table 9: Most active users

userId	number_of_ratings_by_user	avg_rating_by_user
59269	6616	3.264586
67385	6360	3.197720
14463	4648	2.403615
68259	4036	3.576933
27468	4023	3.826871
19635	3771	3.498807
3817	3733	3.112510
63134	3371	3.268170
58357	3361	3.000744
27584	3142	3.001432

Table 10: Least active users

userId	number_of_ratings_by_user	avg_rating_by_user
62516	10	2.250000
22170	12	4.000000

userId	number_of_ratings_by_user	avg_rating_by_user
15719	13	3.769231
50608	13	3.923077
901	14	4.714286
1833	14	3.000000
2476	14	2.928571
5214	14	1.785714
9689	14	3.571429
10364	14	4.321429

From the Figure 8 and the Tables 9-10 we can conclude, that much higher variation among the users who rated less movies, compare to users who rated them a lot and average rating of the users who rated many movies tends to be closer to average (3-3.5).

3.5 Year of release analysis

Year in the title is not useful for analysis and prediction. We need to separate it to own column. Also timestamp as it is is completely uninformative, therefore is being replaced by year, month and day of the week. These operations are performed by the code:

```
edx <- edx %>%
  mutate(year_released = as.numeric(str_sub(title,-5,-2)),
         year_rated = year(as_datetime(timestamp)),
         month_rated = month(as_datetime(timestamp)),
         day_rated = weekdays(as_datetime(timestamp))) %>%
  select(-timestamp)
```

Range of released years in our dataset is from 1915 to 2008. Let's look, how many movies were released by each year and how many ratings they received:

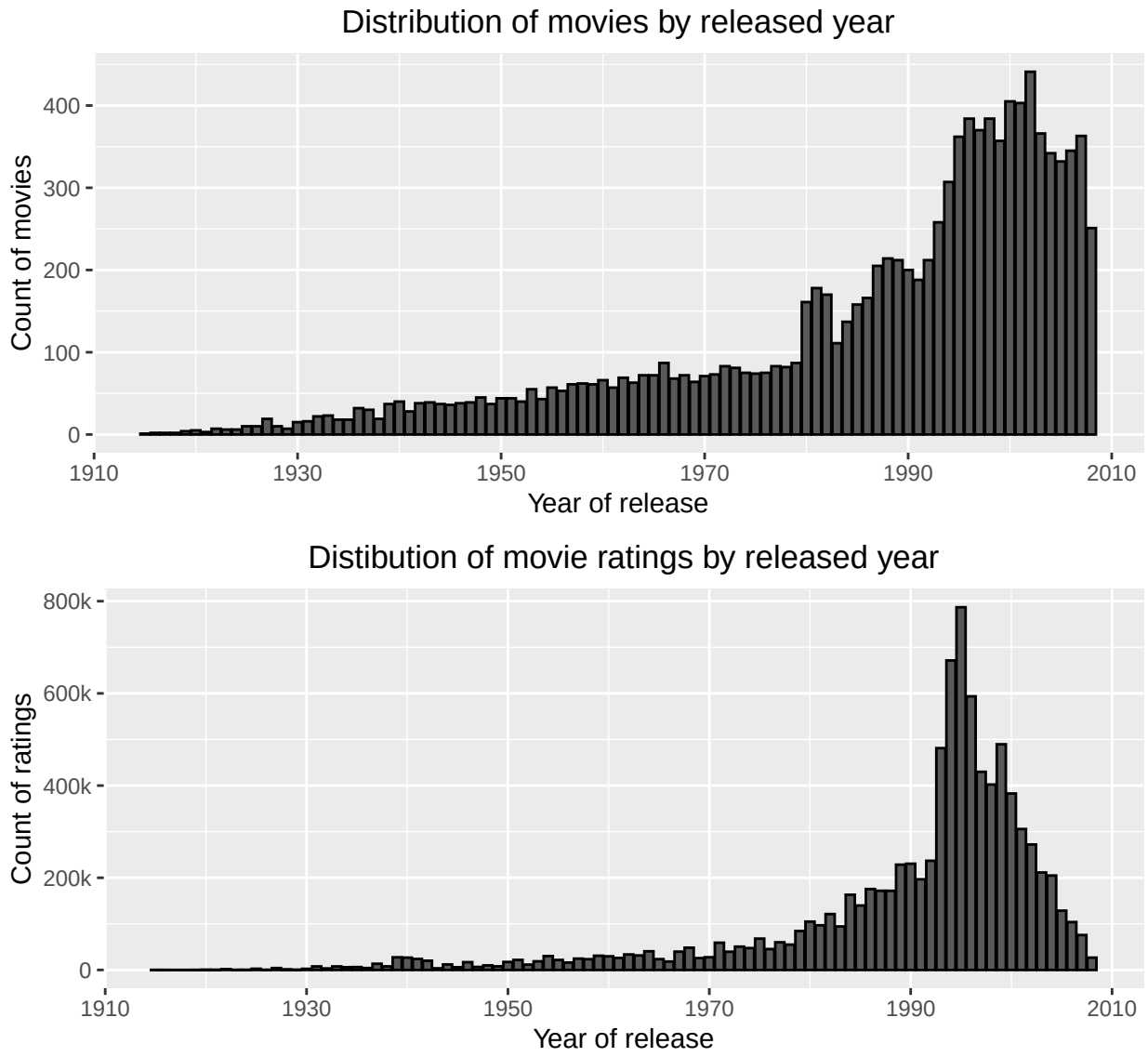

Figure 9: Number of movies and number of movies by year of release

Of course, we are interested about effect of released year on rating:

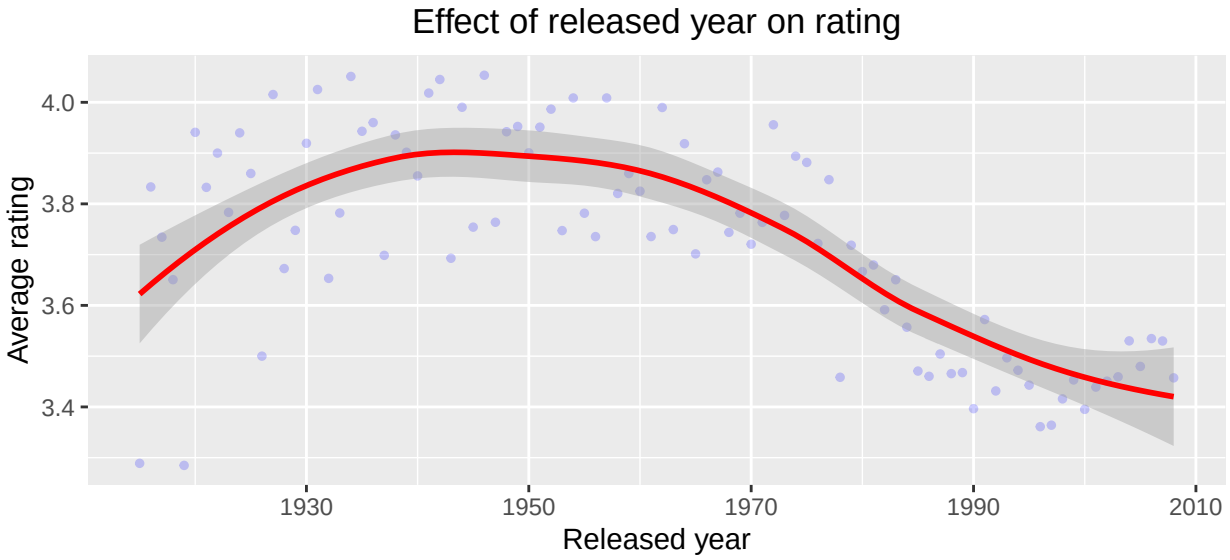

Figure 10: Effect of released year on rating

We see, that at least 250 movies per year are released since 1993. Also, movies of 90-s middle are rated much more often. And movies released in 1940-1960 have higher average rating. It can be explained, that only good movies from that time are still being watched and rated nowadays.

3.6 Rating date analysis

Let's look on effect of rating year on the average rating:

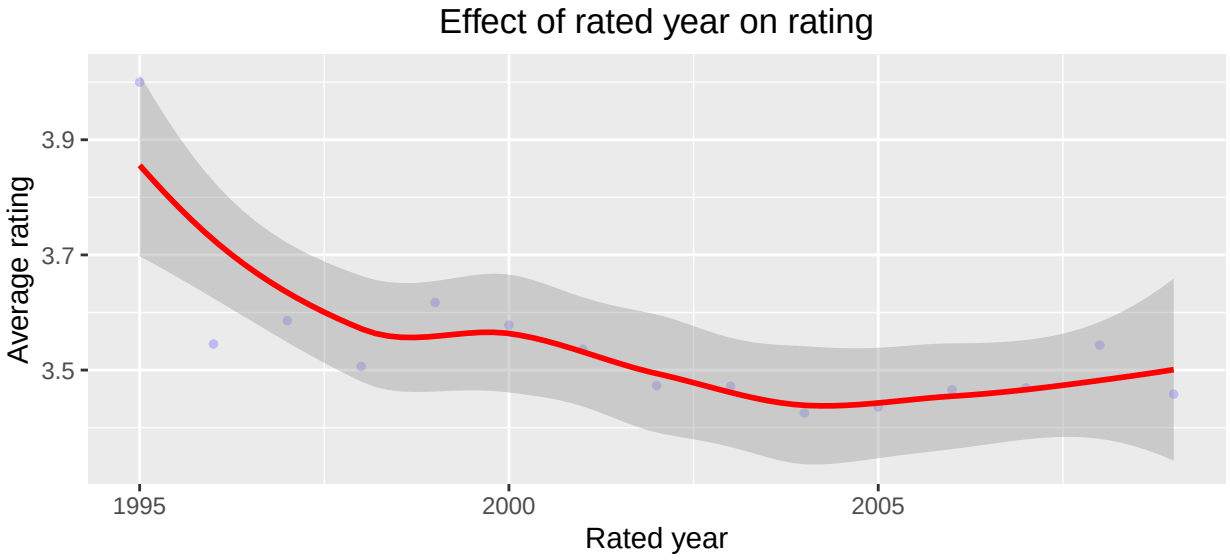

Figure 11: Effect of rated year on rating

First year of rating in our dataset is 1995. Figure 11 shows, that in 1995 users tended to give higher rating to movies than later. It also corresponds to Figure 9: users are watching and rating recent movies more

often, therefore movies released in 90-s have higher average rating.
Effect of rating month on the average rating:

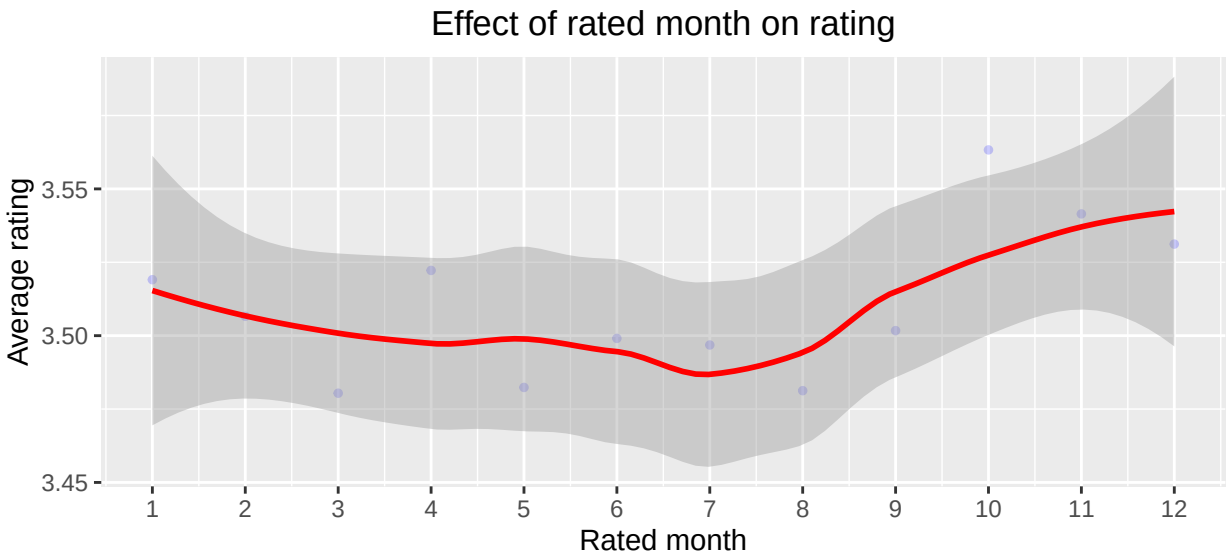

Figure 12: Effect of rated month on rating

Looking on the Figure 12, we can observe that average rating during summer months is lower than across whole year. That can be explained by holiday season and, as consequence, less time spending watching movies. Effect of rating weekday on the average rating:

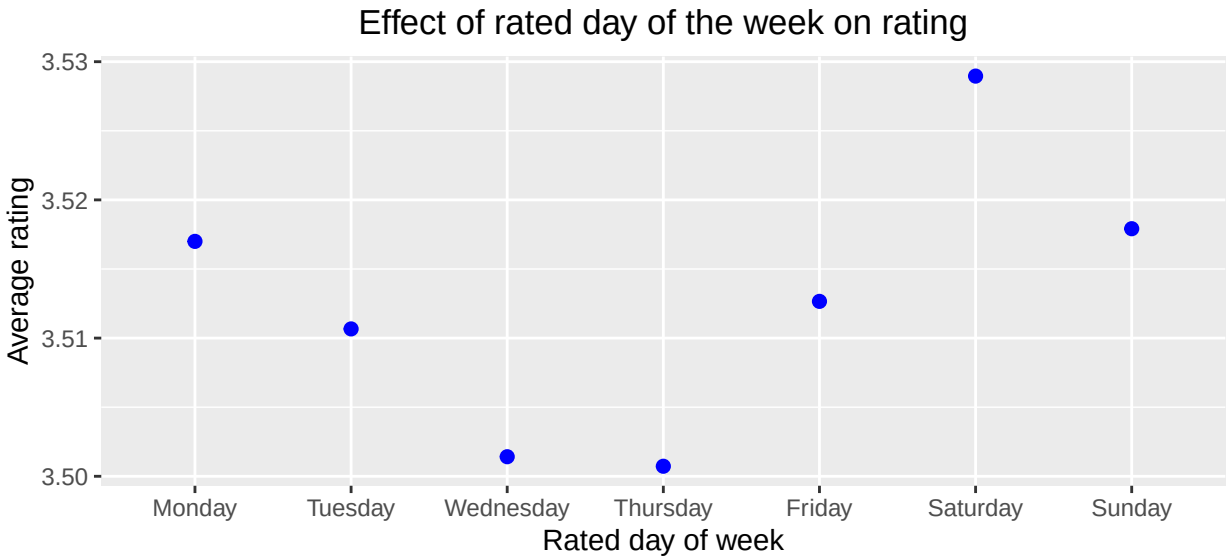

Figure 13: Effect of rated day on rating

Day of week also has some tiny effect on the average rating: a bit higher in weekends. Nevertheless, this difference is very small and we will not use day of week in our model.

3.7 Genres analysis

Each row in the edx dataset contains genres of rated movie. Movie doesn't have to have only one genre, genres combinations are more common (e.g. "Action|War|Drama" and "Action|Comedy" most probably completely different movies, despite both have "Action" in their genres). There are 797 unique genres combinations. Let's look on some of them:

Table 11: Genres combinations overview

x
Comedy|Romance
Action|Crime|Thriller
Action|Drama|Sci-Fi|Thriller
Action|Adventure|Sci-Fi
Action|Adventure|Drama|Sci-Fi
Children|Comedy|Fantasy
Comedy|Drama|Romance|War
Adventure|Children|Romance
Adventure|Animation|Children|Drama|Musical
Action|Comedy

How many of them are unique or about to be unique in our dataset:

Table 12: Unique genres combinations

genres	number_of_ratings
Action|Animation|Comedy|Horror	2
Action|War|Western	2
Adventure|Fantasy|Film-Noir|Mystery|Sci-Fi	2
Adventure|Mystery	2
Crime|Drama|Horror|Sci-Fi	2
Documentary|Romance	2
Drama|Horror|Mystery|Sci-Fi|Thriller	2
Fantasy|Mystery|Sci-Fi|War	2
Action|Adventure|Animation|Comedy|Sci-Fi	3
Horror|War|Western	3
Action|Drama|Horror|Sci-Fi	4
Adventure|Animation|Musical|Sci-Fi	4
Animation|Documentary|War	4
Crime|Documentary	4
Drama|Musical|Thriller	4
Horror|Romance|Thriller	4
Adventure|Horror|Romance|Sci-Fi	5
Comedy|Drama|Horror|Sci-Fi	5
Crime|Drama|Film-Noir|Romance	5
Fantasy|Mystery|Western	5

As we can see, many genres combinations have very few ratings, therefore we will split them to separate genres for further analysis:

```
separated_genres <- edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarise(number_of_ratings = n(), avg_rating = mean(rating))
```

Let's look on separated genres, how often they appear in the dataset and their average ratings:

Table 13: Separated genres

genres	number_of_ratings	avg_rating
Film-Noir	118541	4.011625
Documentary	93066	3.783487
War	511147	3.780813
IMAX	8181	3.767693
Mystery	568332	3.677001
Drama	3910127	3.673131
Crime	1327715	3.665925
(no genres listed)	7	3.642857
Animation	467168	3.600644
Musical	433080	3.563305
Western	189394	3.555918
Romance	1712100	3.553813
Thriller	2325899	3.507676
Fantasy	925637	3.501946
Adventure	1908892	3.493544
Comedy	3540930	3.436908
Action	2560545	3.421405
Children	737994	3.418715
Sci-Fi	1341183	3.395743
Horror	691485	3.269815

Rating of movie can be dependent on it's genre: some genres are more popular than another:

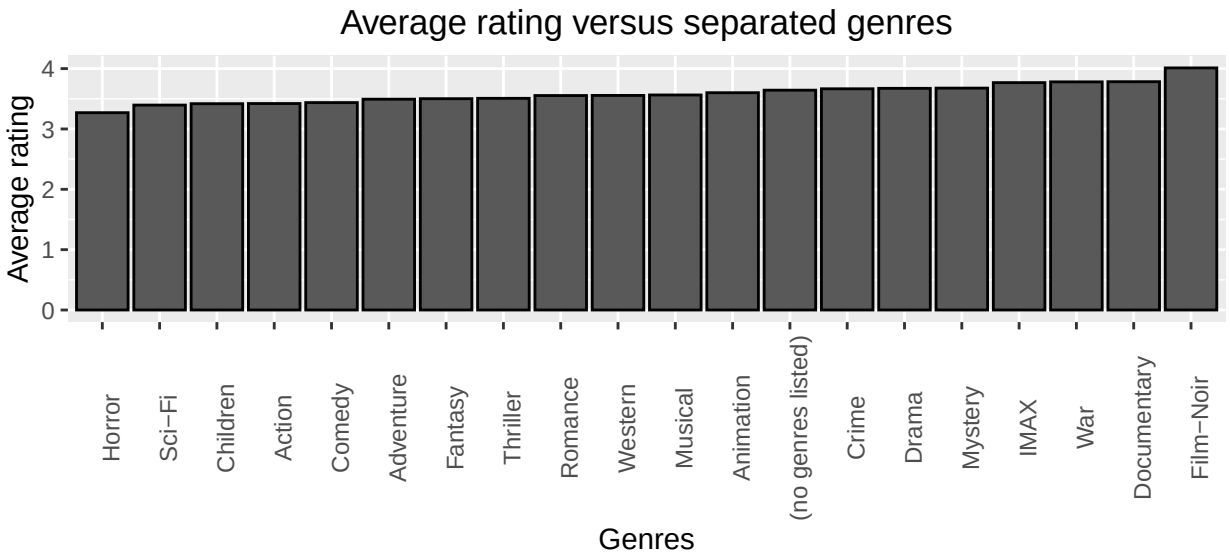

Figure 14: Effect of separated genres on rating

Of course, we should take in account, that there are different number of movies for each genre in our dataset:

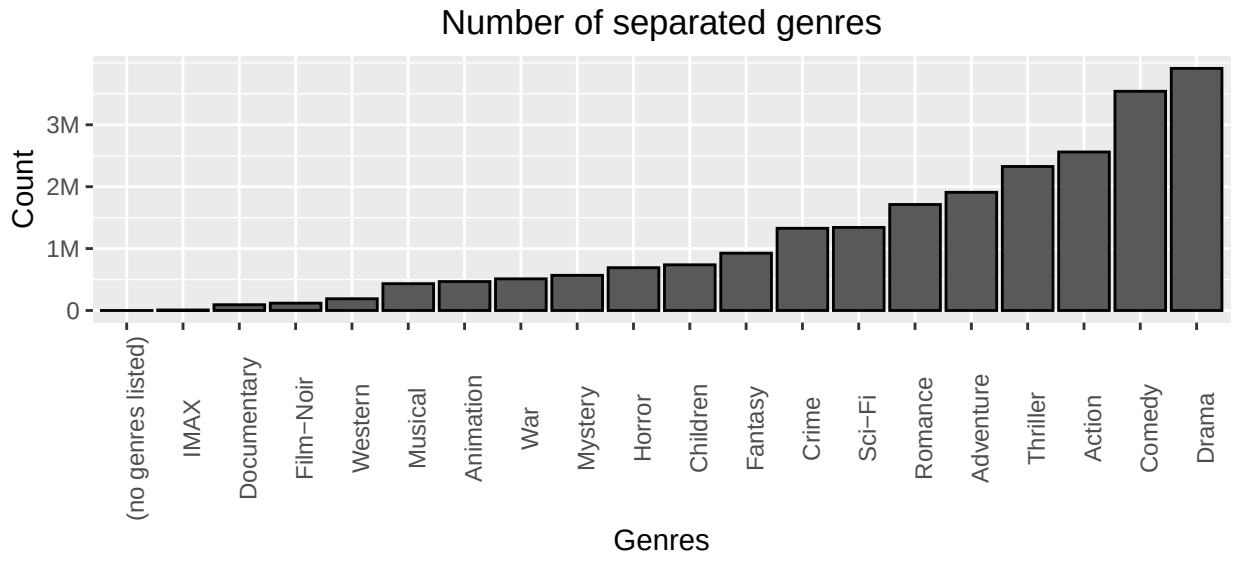

Figure 15: Number of separated genres appearing

We notice, that genre "(no genres listed)" presence in the dataset. Let's check, how many movies have "(no genres listed)":

```
edx %>% filter(genres == "(no genres listed)") %>% group_by(movieId) %>% pull(title) %>% unique()
```

```
## [1] "Pull My Daisy (1958)"
```

Only one movie without genre (from 10677 different movies in the dataset), we can ignore it.
It seems to be clear, that genre affects movie rating. But use 797 different genres combinations, where some of them appear only few times is not convenient. We will check, if we can reduce this amount by finding some strong correlation between genres. E.g. if we see, that fantasy is almost always combined with Sci-Fi, we can keep only fantasy and group similar movies together. We will make a correlation plot, where strong positive correlation between two genres means, that genres very often comes together and negative correlation means that they are almost never combined with each other.

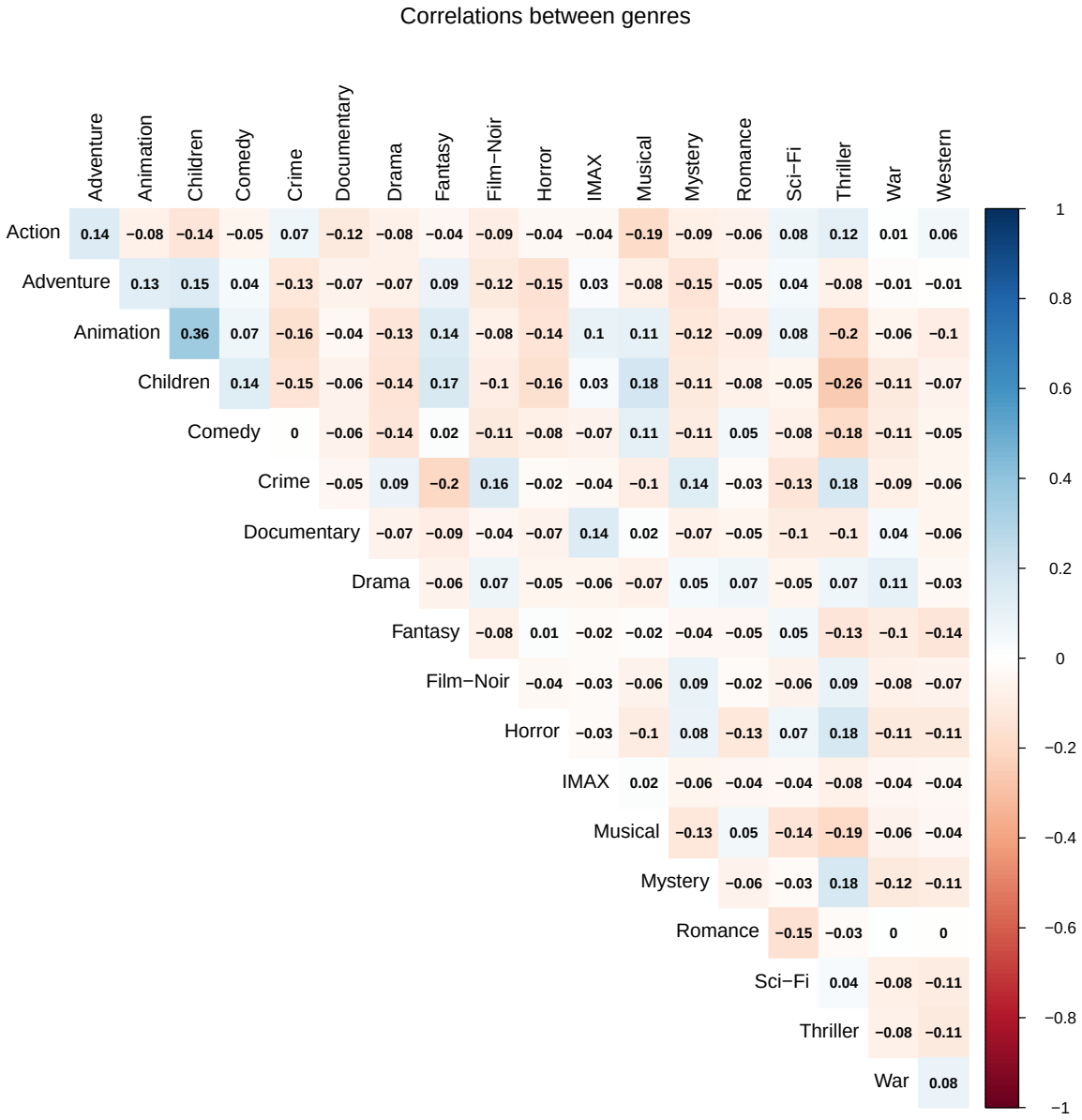

Figure 16: Correlations between genres

Easily, we can see some meaningful correlation only between genres Children and Animation while other genres are relatively independent from each other. We will not reduce amount of genres yet.

3.8 Summary

After performing exploratory data analysis, we can summarize some important observations:

- We have edx dataset, with ratings done by a user for a movie and some information about the movie;
- Ratings are in range 0.5 - 5.0;

- Different average ratings are not equally distributed across movies and users: there are more or less popular movies from one side and more or less cranky users from other side;
- Year of release was combined with movie title in the original dataset, then was separated for analysis;
- Year, month and day of rating were extracted from timestamp column;
- Year of release and genre has an effect on the movie rating;
- Date of rating also has an effect on rating, but usability of this predictor is questionable: do we want to predict movies which user would like at the moment of prediction or we want to just fill historical gaps in matrix from Figure 2? Because first option has clearer appliance, compare to second one, we will try to avoid using rating timestamp in the prediction model;
- Genres of movies are combined differently and there is no strong correlation between multiple genres.

4 Methods of model building

In this chapter we will try different approaches to build prediction model.

4.1 Validation technique

If we train and test the model on the same dataset, we cannot be sure that the same behavior the model will show on real data. Because edx dataset is relatively large, for validation of different models we will use hold-out method: we will split it to train (70%) and test (30%) subsets, both having the same users and movies:

```
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.3, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure userId and movieId in test set are also in train set
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Add rows removed from test set back into train set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

# remove temporary variables
rm(test_index, temp, removed)
```

Check dimensions: dimensions of train-set are 6300114, 9 and dimensions of test-set are 2699941, 9.
Function of the RMSE is defined by code:

```
# function to estimate RMSE
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
  }
```

4.2 First model

In order to have some baseline, we will implement the simplest model. Assume, that the average rating of all movies is the "true" rating, and all variability is explained by random independent error:

$$Y_{u,i} = \mu + \varepsilon_{u,i},$$

where $Y_{u,i}$ is the actual rating, μ is the mean of all ratings, and $\varepsilon_{u,i}$ is the independent errors sampled from the same distribution centered at 0. Compute μ:

```
mu <- mean(train_set$rating)
```

And save RMSE to the list:

```
rmse_results <- data.frame(method = "Just the average", RMSE = RMSE(test_set$rating, mu))
```

With just predicting the average model we have RMSE = 1.0598033. Remember, that RMSE has the same units that the outcome, so we can consider it as just an average error in our predictions. While performing EDA (Chapter 3) we saw, that movies, users, genres and even year of release and date of rating can affect actual rating. We will add different effects to our model.

4.3 Modeling movie effect

We can augment our previous model by adding the term b_i to represent average ranking for movie i:

$$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

It could make sense to perform linear regression on the dataset to find all b_i, but because our dataset is very large, it can take a lot of time and even crash system if computer is not powerful enough. Instead, we will calculate b_i as an average of $Y_{u,i} - \mu$ for each movie i. This will be the least squares estimate $\hat{b}_i$:

```
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

Let's plot movie biases (b_i) and see, how it affects our prediction:

Figure 17: Distribution of movies biases

Figure 17 shows, that large part of ratings variation can be explained by movie effect. It confirms our observation in Chapter 3.3. Let's test our model on a test-set:

```
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i
```

From chapter 3.1 we know, that minimum and maximum ratings in our dataset are 0.5 and 5.0. We will limit our prediction by this range:

```
predicted_ratings <- ifelse(predicted_ratings > 5, 5,
                            ifelse(predicted_ratings < 0.5, 0.5, predicted_ratings))
```

And add the calculated RMSE to the result table:

```
rmse_results <- bind_rows(rmse_results,
                          data.frame(method="Movie effect model",
                                     RMSE = RMSE(predicted_ratings, test_set$rating) ))
```

Current result is:

Table 14: Two models results

method	RMSE
Just the average	1.0598033
Movie effect model	0.9436043

We already have some improvement. But from EDA we know that users also have very strong effect on rating.

4.4 Modeling user + movie effect

We will continue to augment our previous model. Now with user effect b_u. New model is:

$$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$$

```
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

Plot user biases:

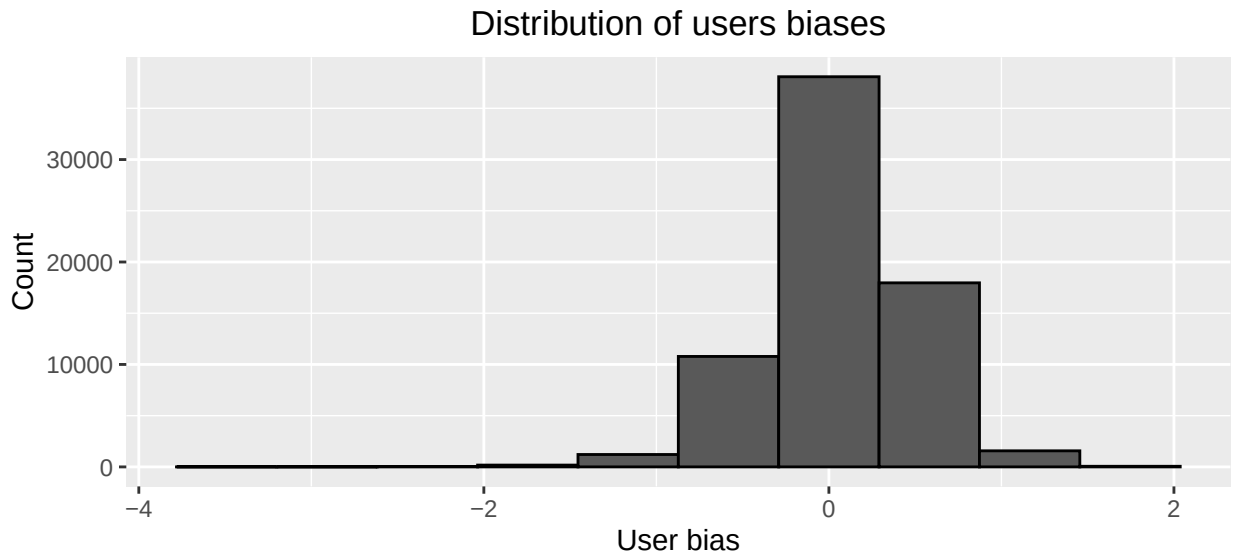

Figure 18: Distribution of users biases

Users biases also have relatively strong effect. Testing the model on the test-set and add the calculated RMSE to the result table:

```
# predict on test-set
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

# limiting rating 0.5 - 5.0, as we know real ratings can't be out of this range
predicted_ratings <- ifelse(predicted_ratings > 5, 5,
                            ifelse(predicted_ratings < 0.5, 0.5, predicted_ratings))

# add calculated RMSE to the table
rmse_results <- bind_rows(rmse_results,
                          data.frame(method="Movie + user effect model",
                                     RMSE = RMSE(predicted_ratings, test_set$rating) ))
```

Current result is:

Table 15: Three models results

method	RMSE
Just the average	1.0598033
Movie effect model	0.9436043
Movie + user effect model	0.8661244

4.5 Modeling user + movie + year of release effects

The same way as we did before, we will add released year effect to our model:

$$Y_{u,i} = \mu + b_i + b_u + b_y + \varepsilon_{u,i},$$

Where b_y is a bias of released year y.

```
year_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(year_released) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u))
```

Plot released year biases:

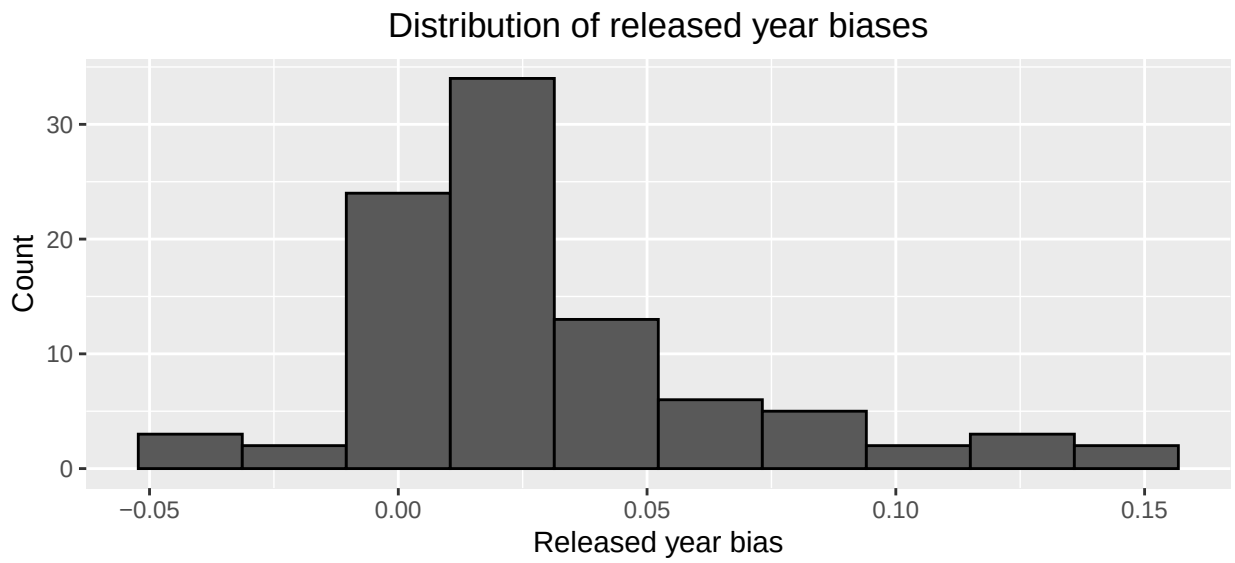

Figure 19: Distribution of released year biases

Effect of released year is not so large, compare to movie or user biases. It looks like we already explained most of ratings variability just with movie and user. Let's see if we improve RMSE on test-set:

```
# predict on test-set
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
```

```
  left_join(user_avgs, by='userId') %>%
  left_join(year_avgs, by='year_released') %>%
  mutate(pred = mu + b_i + b_u + b_y) %>%
  .$pred

# limiting rating 0.5 - 5.0, as we know real ratings can't be out of this range
predicted_ratings <- ifelse(predicted_ratings > 5, 5,
                            ifelse(predicted_ratings < 0.5, 0.5, predicted_ratings))

# add calculated RMSE to the table
rmse_results <- bind_rows(rmse_results,
                          data.frame(method="Movie + user + year effect model",
                                     RMSE = RMSE(predicted_ratings, test_set$rating) ))
```

Current result is:

Table 16: Four models results

method	RMSE
Just the average	1.0598033
Movie effect model	0.9436043
Movie + user effect model	0.8661244
Movie + user + year effect model	0.8658036

Not big improvement. But we still have predictors to add to our model.

4.6 Modeling user + movie + year of release + genre effects

We saw, that different genres attract users differently. Let's compute biases for different genres combinations in our dataset and augment the model with them:

$$Y_{u,i} = \mu + b_i + b_u + b_y + b_g + \varepsilon_{u,i},$$

Where b_g is a bias of genres combination g.

```
genre_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avgs, by='year_released') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u - b_y))
```

Plot genres biases:

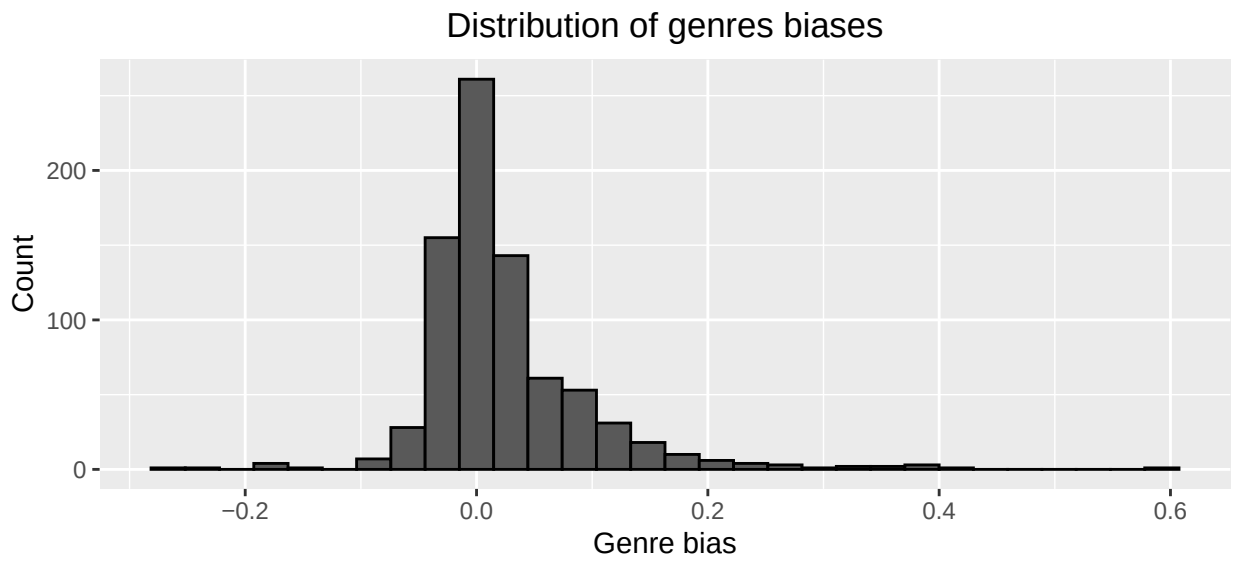

Figure 20: Distribution of released year biases

Test-set validation:

```
# predict on test-set
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avgs, by='year_released') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
  .$pred

# limiting rating 0.5 - 5.0, as we know real ratings can't be out of this range
predicted_ratings <- ifelse(predicted_ratings > 5, 5,
                            ifelse(predicted_ratings < 0.5, 0.5, predicted_ratings))

# add calculated RMSE to the table
rmse_results <- bind_rows(rmse_results,
                          data.frame(method="Movie + user + year + genre Effect Model",
                                     RMSE = RMSE(predicted_ratings, test_set$rating) ))
```

Current result is:

Table 17: Five models results

method	RMSE
Just the average	1.0598033
Movie effect model	0.9436043
Movie + user effect model	0.8661244
Movie + user + year effect model	0.8658036
Movie + user + year + genre Effect Model	0.8655260

Again, not really big improvement and less and less variation can be explained by just adding new predictors. Let's try to optimize our model.

4.7 Regularization model

4.7.1 Four predictors

Remember (Chapter 3.3 and 3.4) that some movies were rated just few times and some users rated only few movies. We need to regularize our model by implementing penalty large estimates which are formed using small sample sizes. In other words, we want to constrain the total variability of the effect sizes by penalizing large estimates that come from small sample sizes. To meet our goal, instead of minimizing the least squares equation $y_{u,i} - \mu - b_i - b_u - b_y - b_g$, we minimize an equation that adds a penalty:

$$\sum_{u,i}(y_{u,i} - \mu - b_i - b_u - b_y - b_g)^2 + \lambda(\sum_i b_i^2 + \sum_u b_u^2 + \sum_y b_y^2 + \sum_g b_g^2)$$

To find lambda, which minimize the error, we will use test-train set cross-validation:

```
# find the best lambda (small sample size penalty)
lambdas <- seq(0, 10, 0.25)

# build models for different lambdas
# train-test set cross validation is used, because dataset is relatively big
rmses <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  b_y <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(year_released) %>%
    summarize(b_y = sum(rating - b_i - b_u - mu)/(n()+l))

  b_g <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by="year_released") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - b_y - mu)/(n()+l))

  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "year_released") %>%
```

```
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})

lambda <- lambdas[which.min(rmses)]
```

We can plot all lambdas vs. RMSE (cross-validation result):

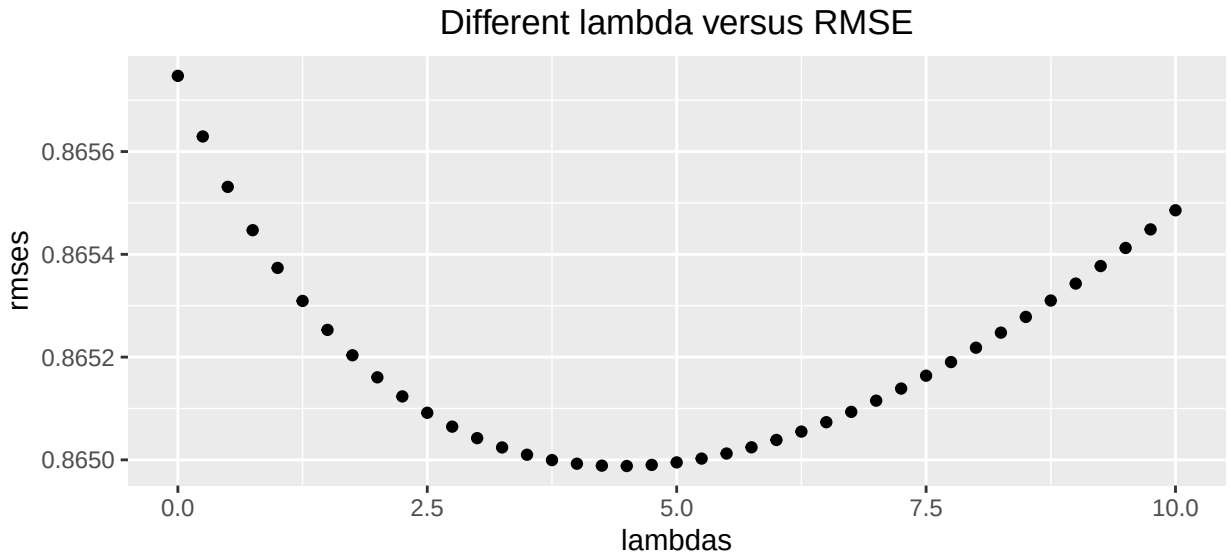

Figure 21: Different lambda versus RMSE

The best lambda is 4.5. Let's rebuild the same model, but with regularization with the best lambda penalty and get RMSE on test-set:

```
# add regularized model with the best lambda
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))


b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

b_y <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(year_released) %>%
  summarize(b_y = sum(rating - b_i - b_u - mu)/(n()+lambda))
```

```
b_g <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_released") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - b_y - mu)/(n()+lambda))

# predict on test-set
predicted_ratings <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_y, by = "year_released") %>%
  left_join(b_g, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
  .$pred

# limiting rating 0.5 - 5.0, as we know real ratings can't be out of this range
predicted_ratings <- ifelse(predicted_ratings > 5, 5,
                            ifelse(predicted_ratings < 0.5, 0.5, predicted_ratings))

# add calculated RMSE to the table
rmse_results <- bind_rows(rmse_results,
              data.frame(method="Regularized movie + user + year + genre effect model",
                         RMSE = RMSE(predicted_ratings, test_set$rating) ))
```

Result is:

Table 18: Six models results

method	RMSE
Just the average	1.0598033
Movie effect model	0.9436043
Movie + user effect model	0.8661244
Movie + user + year effect model	0.8658036
Movie + user + year + genre Effect Model	0.8655260
Regularized movie + user + year + genre effect model	0.8648679

Better result, and meet our goal: RMSE < 0.86490. But there is still room to improve.

4.7.2 Six predictors

Let's try to add more predictors and regularize their biases as well. We saw, that year and month of rating also affects the rating. As was mentioned in the Chapter 3.8, using of these predictors is questionable, because it limits practical application of our model: it doesn't make sense to suggest to some user **u** to watch some movie **i** five years ago. But let's see, how using year of rating and month of rating can improve our model. We will repeat the same code that we used above to build regularized model on movie, user, released year and rating, just add year and month of rating in it. First, find the best lambda:

```
# regularization of all predictors
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
```

```
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  b_y <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(year_released) %>%
    summarize(b_y = sum(rating - b_i - b_u - mu)/(n()+l))

  b_g <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by="year_released") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - b_y - mu)/(n()+l))

  b_ry <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by="year_released") %>%
    left_join(b_g, by="genres") %>%
    group_by(year_rated) %>%
    summarize(b_ry = sum(rating - b_i - b_u - b_y - b_g - mu)/(n()+l))

  b_rm <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by="year_released") %>%
    left_join(b_g, by="genres") %>%
    left_join(b_ry, by="year_rated") %>%
    group_by(month_rated) %>%
    summarize(b_rm = sum(rating - b_i - b_u - b_y - b_g - b_ry - mu)/(n()+l))


  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "year_released") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_ry, by = "year_rated") %>%
    left_join(b_rm, by = "month_rated") %>%
    mutate(pred = mu + b_i + b_u + b_y + b_g + b_ry + b_rm) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})
```

```
lambda <- lambdas[which.min(rmses)]
```

Plot all lambdas vs. RMSE (cross-validation result):

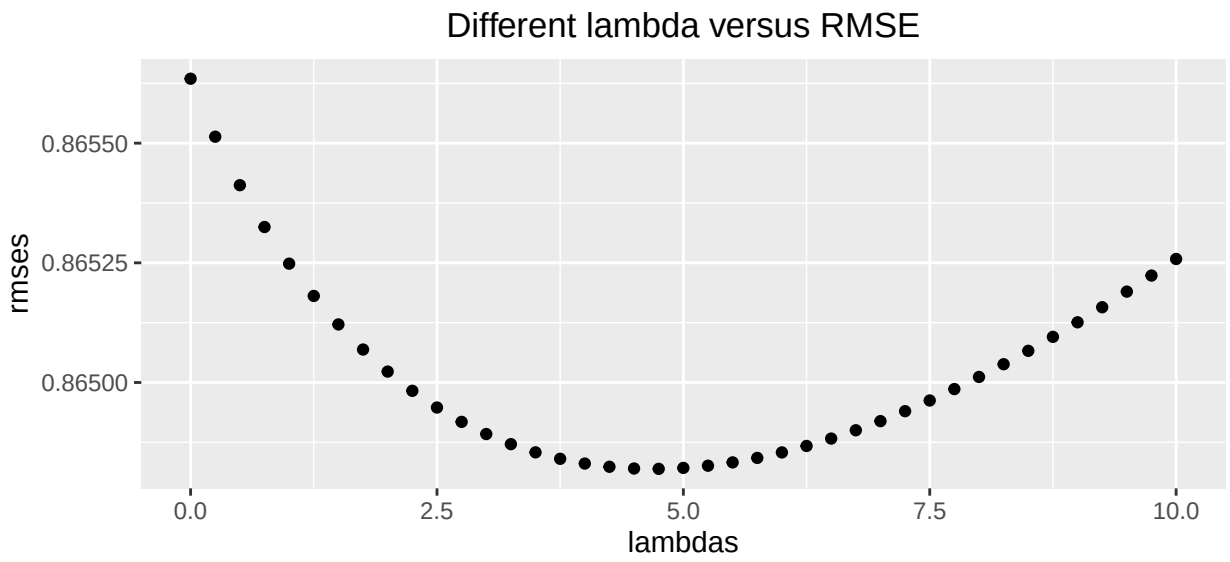

Figure 22: Different lambda versus RMSE

The best lambda is 4.75. Let's rebuild the regularized model with all predictors and best lambda:

```
# add regularized model with the best lambda
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))


b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

b_y <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(year_released) %>%
  summarize(b_y = sum(rating - b_i - b_u - mu)/(n()+lambda))

b_g <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_released") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - b_y - mu)/(n()+lambda))

b_ry <- train_set %>%
```

```
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_released") %>%
  left_join(b_g, by="genres") %>%
  group_by(year_rated) %>%
  summarize(b_ry = sum(rating - b_i - b_u - b_y - b_g - mu)/(n()+lambda))

b_rm <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_released") %>%
  left_join(b_g, by="genres") %>%
  left_join(b_ry, by="year_rated") %>%
  group_by(month_rated) %>%
  summarize(b_rm = sum(rating - b_i - b_u - b_y - b_g - b_ry - mu)/(n()+lambda))

# predict on test-set
predicted_ratings <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_y, by = "year_released") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_ry, by = "year_rated") %>%
  left_join(b_rm, by = "month_rated") %>%
  mutate(pred = mu + b_i + b_u + b_y + b_g + b_ry + b_rm) %>%
  pull(pred)

# limiting rating 0.5 - 5.0, as we know real ratings can't be out of this range
predicted_ratings <- ifelse(predicted_ratings > 5, 5,
                            ifelse(predicted_ratings < 0.5, 0.5, predicted_ratings))

# add calculated RMSE to the table
rmse_results <- bind_rows(rmse_results,
data.frame(method="Regularized movie + user + year + genre + rating year and month effect model",
            RMSE = RMSE(predicted_ratings, test_set$rating) ))
```

Result is:

Table 19: Seven models results

method	RMSE
Just the average	1.0598033
Movie effect model	0.9436043
Movie + user effect model	0.8661244
Movie + user + year effect model	0.8658036
Movie + user + year + genre Effect Model	0.8655260
Regularized movie + user + year + genre effect model	0.8648679
Regularized movie + user + year + genre + rating year and month effect model	0.8646965

Little improvement is achieved, by adding rating year and month. To get some more meaningful improvement we will try another technique.

4.8 Matrix factorization

4.8.1 Principal Component Analysis

We saw, that the movie and the user have the strongest effect on rating. Indeed, different groups of users like different kinds of movies. We could group users by age, sex, nationality, but 1) we don't have such information in our dataset, 2) it will not explain variability completely. Grouping movies by genre also is not very useful: if one likes action/sci-fi, it doesn't mean that he likes all movies which have these genres in the description. Instead, we will try to find patterns, how different users rated different movies and make our prediction based on this pattern. To reduce dimensions of our model, we will use only userId and movieId. Year/month of rating also to be out of this model, which give us possibility to use it as real recommendation system "online". Ratings patterns can be discovered by studying the residuals:

$$r_{u,i} = y_{u,i} - \hat{b}_i - \hat{b}_u$$

We will convert the data into a matrix so that each user gets a row, each movie gets a column, and $y_{u,i}$ is the entry in row u and column i. Because this operation performed on large dataset takes too much time, we will use small subset, with movies which were rated 3000 or more times and users who rated 250 or more movies:

```
train_small <- train_set %>%
  group_by(movieId) %>%
  filter(n() >= 3000) %>% ungroup() %>%
  group_by(userId) %>%
  filter(n() >= 250) %>% ungroup()

# convert it to matrix with users in rows, movies in columns ans ratings in cells
y <- train_small %>%
  select(userId, movieId, rating) %>%
  pivot_wider(names_from = "movieId", values_from = "rating") %>%
  as.matrix()


# add row names and column names
rownames(y)<- y[,1]
y <- y[,-1]


# name columns as movie title, instead of movieId
movie_titles <- train_set %>%
  select(movieId, title) %>%
  distinct()

# apply to columns
colnames(y) <- with(movie_titles, title[match(colnames(y), movieId)])


# convert them to residuals by removing the column and row effects
y <- sweep(y, 2, colMeans(y, na.rm=TRUE))
y <- sweep(y, 1, rowMeans(y, na.rm=TRUE))
```

As we remember (Chapter 3.2), not each user rated each movie. To have our matrix filled completely we can use Principal Component Analysis. Principal component analysis (PCA) is a statistical procedure that allows you to summarize the information content in large data tables by means of a smaller set of "summary

indices" that can be more easily visualized and analyzed. To impute our users/movies matrix with missing value we will use *imputePCA()* function from "missMDA" package.

```
pca_md <- imputePCA(y, ncp = 3)
y_md <- pca_md$completeObs
```

And then we will apply *prcomp()* function, from default R "stats" package, on the imputed matrix in order to perform PCA and find all principal components:

```
pca <- prcomp(y_md)
```

Now, instead of two predictors we have multiple Principal Components (PC), which are able to explain outcomes variability. Let's plot them and see, how valuable they can be:

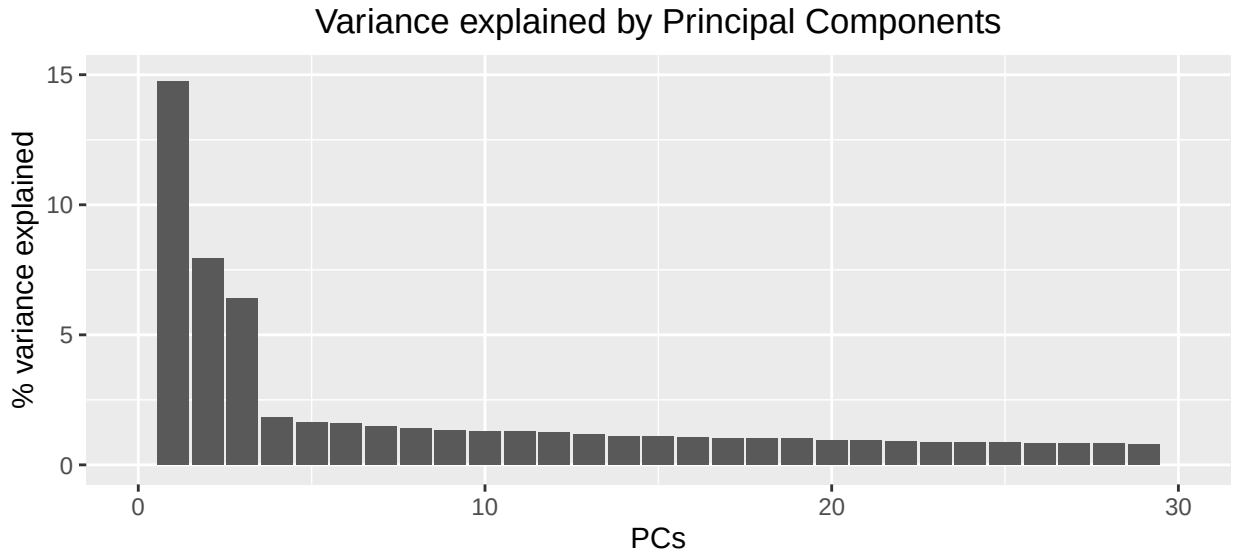

Figure 23: Variance explained by Principal Components

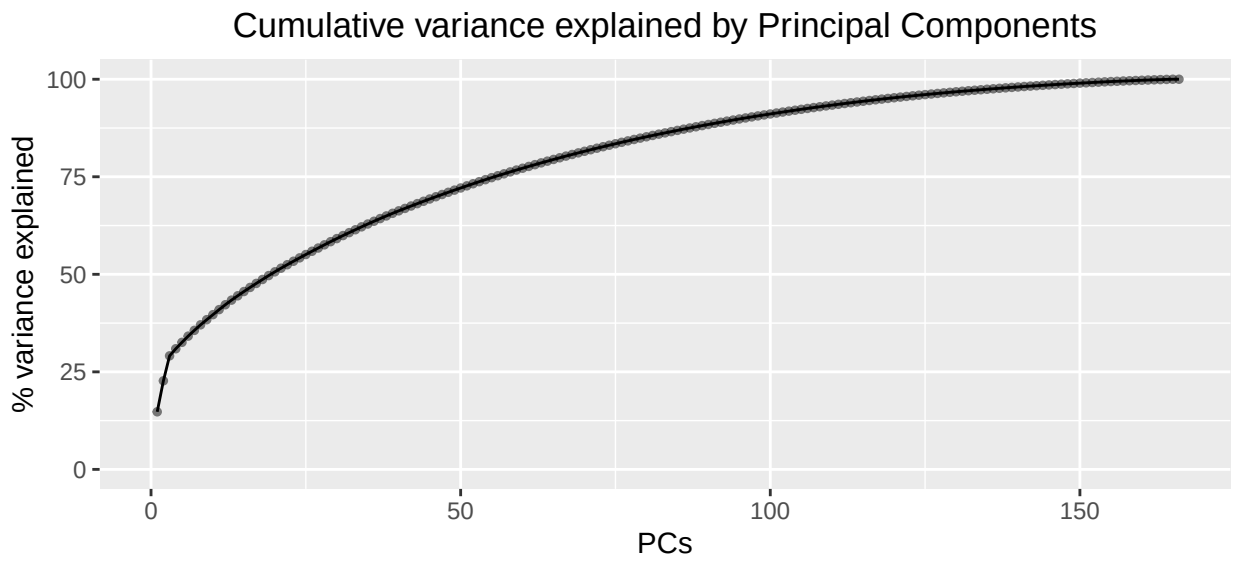

Figure 24: Cumulative variance explained by Principal Components

Looks like we can explain a lot of variability by just several PCs. We can visualize model structure of two first principal components for some well-known movies:

```
sample_movies <- c("Bourne", "Star Trek", "Truman Show", "Beauty and the Beast",
                   "Pulp Fiction", "Matrix", "Harry Potter", "Goldfinger",
                   "Mary Poppins", "Jumanji",
                   "Shawshank Redemption", "Piano", "Trainspotting",
                   "Birds", "Toy Story", "Contact")
```

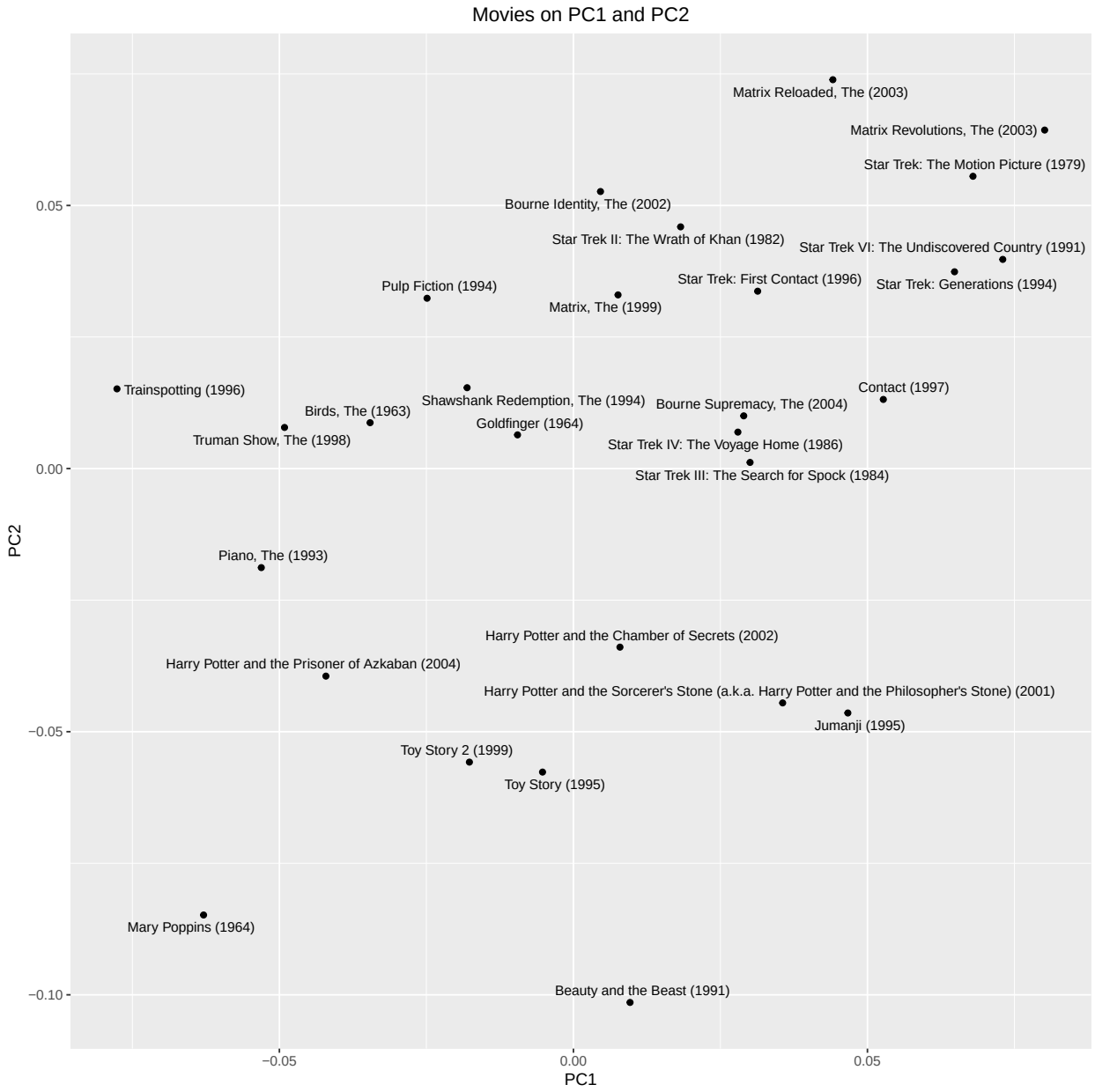

Figure 25: Variance explained by Principal Components

From the Figure 25 we can see we can see some strong correlations between some movies. That makes sense, that people who love Matrix also like Start Trek, e.g. Studying horizontal and vertical axes, we see that PC1 is responsible for Sci-Fi/Fantasy vs. Drama movies and PC2 shows children movies vs. adult movies. Here is explained the concept behind matrix factorization. We try to predict the ratings by searching for similarities between users and similarities between movies.

4.8.2 Recosystem model

Most of computer are not able to perform PCA on our complete dataset - it is too large. Fortunately, there is a "recosystem" package for R, which is the wrapper of the 'libmf' library. LIBMF is an open source tool

for approximating an incomplete matrix using the product of two matrices in a latent space. Main features of LIBMF include:

- providing solvers for real-valued matrix factorization, binary matrix factorization, and one-class matrix factorization;
- parallel computation in a multi-core machine;
- using CPU instructions (e.g., SSE) to accelerate vector operations;
- taking less than 20 minutes to converge to a reasonable level on a data set of 1.7B ratings;
- cross validation for parameter selection;
- supporting disk-level training, which largely reduces the memory usage

Using it we can solve our task based only by userId and MovieId and because it uses low-level CPU instruction, we can build a model on an average laptop with an acceptable time.

First, let's try to build the model with default parametres, for first estimation:

```
# create new model object
r = Reco()

# keep only userId, movieId (predictors) and rating (outcome) from the trainset
reco_train <- train_set %>%
  select(userId, movieId, rating)

# create an object of class "DataSource" from the new trainset,
# to use it in train(), tune() and predict() functions
reco_train <- with(reco_train,
                   data_memory(user_index = userId, item_index = movieId,
                               rating = rating, index1 = TRUE))

# train model with default parameters (first check)
r$train(reco_train, opts = c(niter = 20, verbose = FALSE))

# predict on the testset
# keep only predictors: userId and movieId
reco_test <- test_set %>%
  select(userId, movieId)

# convert to DataSource
reco_test <- with(reco_test,
                  data_memory(user_index = userId, item_index = movieId, index1 = TRUE))

# predict
predicted_ratings <- r$predict(reco_test)

# check RMSE
RMSE(predicted_ratings, test_set$rating)
```

```
## [1] 0.8346417
```

We got RMSE 0.8346417 with just default parameters and 20 iterations. Most probably we can improve our result. Because recosystem has k-fold cross-validation integrated in *tune()* function, we will use 3-fold cross-validation to find best model parameters:

```
opts = r$tune(reco_train,
              opts = list(dim = c(5, 10, 20, 50),
                          lrate = c(0.1, 0.2),
                          costp_l1 = c(0), costq_l1 = c(0),
                          costp_l2 = c(0, 0.01, 0.1, 0.3),
                          nthread = 4, niter = 20, nfold = 3))
best_options <- opts$min
```

Now, let's train the model with the best parameters on the train-set and test it on test-set:

```
# train model with the best parameters
r$train(reco_train, opts = c(best_options, niter = 50, verbose = FALSE))

# predict on test_set
predicted_ratings <- r$predict(reco_test)

# limiting rating 0.5 - 5.0, as we know real ratings can't be out of this range
predicted_ratings <- ifelse(predicted_ratings > 5, 5,
                            ifelse(predicted_ratings < 0.5, 0.5, predicted_ratings))

# add calculated RMSE to the table
rmse_results <- bind_rows(rmse_results,
                          data.frame(method="Recosystem matrix factorization model",
                                     RMSE = RMSE(predicted_ratings, test_set$rating) ))
```

Our results are:

Table 20: Eight models results

method	RMSE
Just the average	1.0598033
Movie effect model	0.9436043
Movie + user effect model	0.8661244
Movie + user + year effect model	0.8658036
Movie + user + year + genre Effect Model	0.8655260
Regularized movie + user + year + genre effect model	0.8648679
Regularized movie + user + year + genre + rating year and month effect model	0.8646965
Recosystem matrix factorization model	0.7979562

It is much better than previous models.

The latest model shows very good result. Further improvement is beyond the capstone project, therefore this model will be selected as the final one. Last, we will train the model on the complete edx dataset:

```
# create new model object
r = Reco()

# keep only userId, movieId (predictors) and rating (outcome) from the edx
```

```
reco_edx <- edx %>%
  select(userId, movieId, rating)

# convert to recosystem DataSource object
reco_edx <- with(reco_edx,
                   data_memory(user_index = userId, item_index = movieId,
                               rating = rating, index1 = TRUE))

# train with selected by cross-validation best parameters on a complete edx dataset
r$train(reco_edx, opts = c(best_options, niter = 50, verbose = FALSE))
```

r is the our final model.

5 Validation result

We didn't use **validation** set during complete analysis. All model selection, training, cross-validation were done purely using **edx** set. Now it is time to test our final model on the hold-out validation set:

```
# keep only userId and movieId in validation set
reco_validation <- validation %>%
  select(userId, movieId)

# specify source for recommender system
reco_validation <- with(reco_validation,
                 data_memory(user_index = userId, item_index = movieId, index1 = TRUE))

# predict using our best model
validation_predict <- r$predict(reco_validation)

# limit predictions in range 0.5 - 5.0
validation_predict <- ifelse(validation_predict > 5, 5,
                             ifelse(validation_predict < 0.5, 0.5, validation_predict))

model_validation_rmse <- RMSE(validation_predict, validation$rating)
rmse_results <- bind_rows(rmse_results,
                data.frame(method="Final model on validation (final hold-out) set",
                           RMSE = model_validation_rmse ))
```

Our final RMSE on validation dataset is: 0.784991. All models results:

Table 21: Final models results

method	RMSE
Just the average	1.0598033
Movie effect model	0.9436043
Movie + user effect model	0.8661244
Movie + user + year effect model	0.8658036
Movie + user + year + genre Effect Model	0.8655260
Regularized movie + user + year + genre effect model	0.8648679
Regularized movie + user + year + genre + rating year and month effect model	0.8646965
Recosystem matrix factorization model	0.7979562

method	RMSE
Final model on validation (final hold-out) set	0.7849910

6 Conclusion

In this project we have built a recommendation system based on MovieLens dataset. We performed exploratory data analysis on the dataset, tested different approaches for outcomes prediction. Our final model is based on matrix factorization and principal component analysis.
At the end, the model was tested on the hold-out dataset and the model reported root square mean error 0.784991, this is lower than project requirement 0.86490. Rating timestamp is not used in the model, so the model can be used for "online" recommendations.

Possible future development of the model can be:

- Gathering additional information about users to predict their movie tastes;
- Instead of trying to predict exact rating, try to predict if user **u** would like movie **i** or not;
- Use deep learning algorithms;
- Periodically retrain model using recent users true rating, because users can change their preferences over the time. Also, new users and movies must be added to the dataset.

7 Literature

1. Francesco Ricci and Lior Rokach and Bracha Shapira, Introduction to Recommender Systems Handbook
2. Rafael A. Irizarry, Introduction to Data Science
3. Documentation to 'recosystem' package
4. Ian T. Jolliffe and Jorge Cadima, Principal component analysis: a review and recent developments